

# ***Secure SaaS Supply Chains (SBOM & NLP Integration)***

**Hackathon Challenge:** "Leveraging SPARQL, Flask, and RDF for Software Component Analysis"

**Team:** Megan Mashburn, Yaser Shafik, Ankita Patel, Ibrahim Olanigan, and Gerry Murphy

## The Challenge of SBOM Management (Pain Points)

### Integration with Knowledge Graph:

**Challenge:** Integrating Apache Jena for SPARQL queries was complex and time-consuming.

- **Impact:** Required extensive configuration to ensure correct backend connectivity and optimize query performance.

### Large Dataset:

**Challenge:** Filtering by software/hardware name returned too many results, making it hard to focus on specific data.

- **Impact:** Slowed down query performance and increased time spent fine-tuning filters.

### Excessive Query Results:

**Challenge:** Even with specific version filters, results were still overwhelming.

- **Impact:** Required disabling result limiters, complicating query performance and response time.

### Data Loss in SPARQL Queries:

**Challenge:** Some SPARQL queries caused data loss due to graph structure.

- **Impact:** Introduced gaps in the SBOM, affecting data completeness and accuracy.

### Conflicting Vulnerability Data:

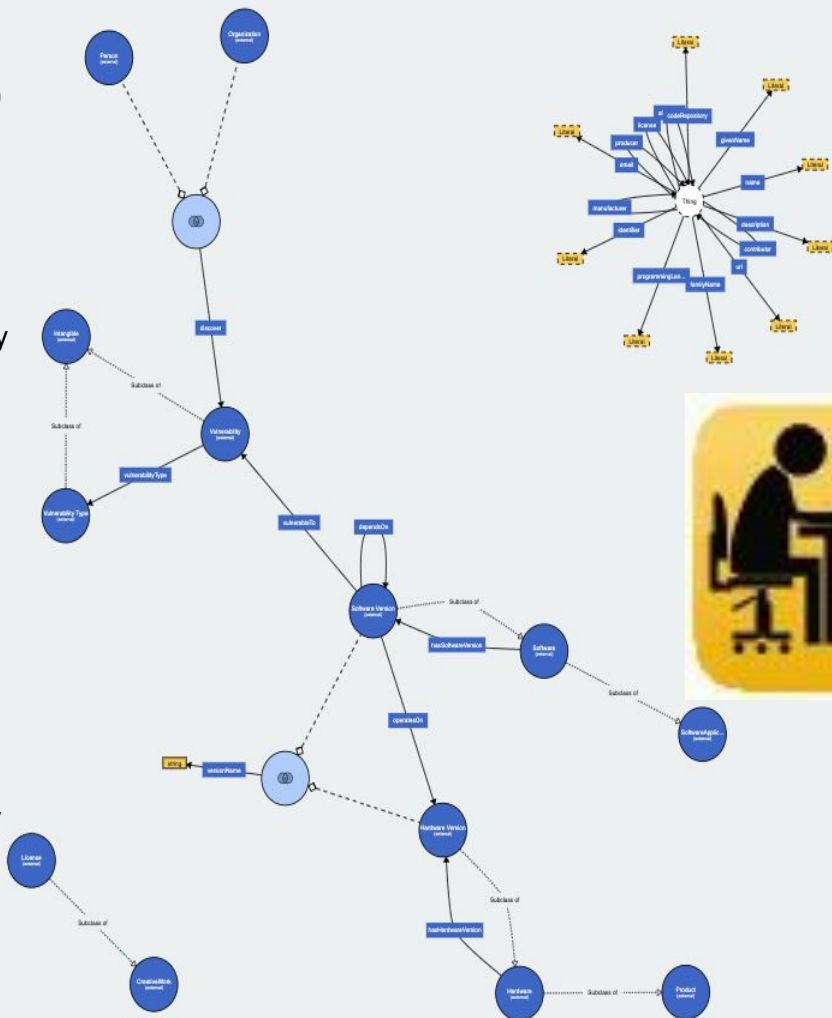
**Challenge:** Versions marked as vulnerable sometimes returned as "safe."

- **Impact:** Created uncertainty in the accuracy of vulnerability assessments.

### Mismatched Data from Multiple Sources:

**Challenge:** Subject lines from API responses didn't match product/version data.

- **Impact:** Required additional cleanup to remove "filler" data and ensure correct mappings.



# Introducing SBOM Analyzer (Solution)

[Return Home](#)

## SBOM Details

Product Name

Python

[Get SBOM Data](#)

### SBOM Data for Python

Software Version: python \*

Dependency On: condor 7.8.2~dfsg.1-1+deb7u3

Vulnerability: CVE-2012-5197

Software Version: python \*

### Safe Python Versions

Recommended Version: python 3

Recommended Version: python \*

Recommended Version: python >=2.7.1-0ubuntu2

## Integration with Knowledge Graph:

**Solution:** Integrated Apache Jena with SPARQL queries via Flask and SPARQLClient to retrieve data from the knowledge graph.

- **Result:** Reduced setup complexity and ensured smooth integration for querying, optimizing performance.

## Large Dataset:

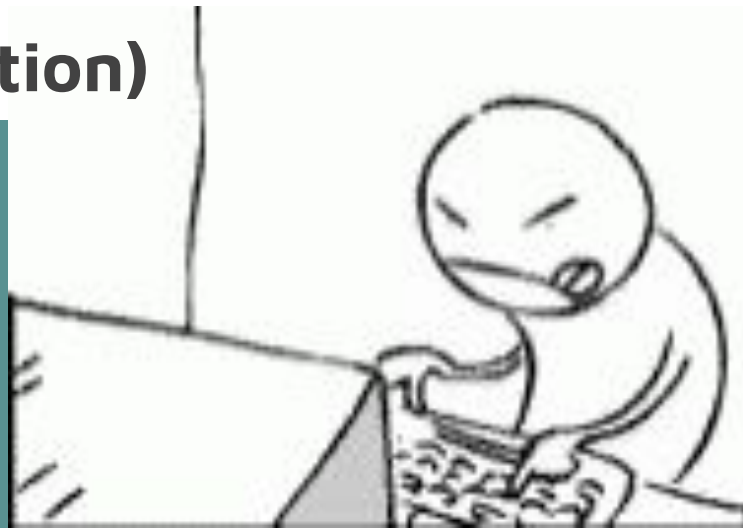
**Solution:** Implemented indexed filtering and pagination techniques to manage large datasets and return manageable results.

- **Result:** Improved query performance and responsiveness by breaking down results into smaller, more focused chunks.

## Excessive Query Results

**Solution:** Used dynamic query optimization with filters for versioning and dependency checks to manage large results.

- **Result:** Refined SPARQL queries to include only necessary data, limiting returned results by filtering specific product versions and dependencies.



```
{sbomData.length > 0 && (
  <div className="scrollable-data">
    <h3>SBOM Data for {productName}</h3>
    {sbomData.map((item, index) => (
      <div key={index} className="sbom-item">
        <p><strong>Software Version:</strong> {item.softwareVersion}</p>
        <p><strong>Dependency On:</strong> {item.dependency}</p>
        <p><strong>Vulnerability:</strong> {item.vulnerability}</p>
        <hr />
      </div>
    ))}
  </div>
)}

{recData.length > 0 && (
  <div className="scrollable-data">
    <h3>Safe {productName} Versions</h3>
    {recData.map((item, index) => (
      <div key={index} className="recommendation-item">
        <p><strong>Recommended Version:</strong> {item.softwareVersion}</p>
      </div>
    ))}
  </div>
)}
)}
```

# Introducing SBOM Analyzer

## (Solutions 2)

```
def get_details(self, product_name):
    try:
        logging.info(f"Generating full SBOM for {product_name}")
        query = f"""
        PREFIX sc: <https://w3id.org/secure-chain/>
        PREFIX schema: <http://schema.org/>

        SELECT ?version ?dependency ?cve
        WHERE {{
            ?lib schema:name "{product_name}" .
            ?lib sc:hasSoftwareVersion ?version .
            ?dependency sc:dependsOn+ ?version .
            ?dependency sc:vulnerableTo ?cve .
        }}
        """
        logging.debug(f"Generated query: {query}")
        results = self.sparql_client.query(query)
        if results and 'results' in results and 'bindings' in results['results']:
            sbom_data = []
            for binding in results['results']['bindings']:
                dependency_uri = binding.get('dependency', {}).get('value', '')
                version_uri = binding.get('version', {}).get('value', '')
                cve_uri = binding.get('cve', {}).get('value', '')
                sbom_data.append({
                    'dependency': self.format_dependency(dependency_uri),
                    'softwareVersion': self.format_dependency(version_uri),
                    'vulnerability': self.format_vulnerability(cve_uri),
                })
            return sbom_data
        else:
            logging.debug("No SBOM found.")
            return []

    except Exception as e:
        logging.error(f"Error generating SBOM: {e}", exc_info=True)
        return "Hey nothing here"
```

### Data Loss in SPARQL Queries

**Solution:** Added enhanced error handling and logging to track and resolve any issues in fetching data, ensuring that all relevant dependencies and vulnerabilities were captured.

- **Result:** Ensured full data capture by refining and testing query logic.

### Conflicting Vulnerability Data

**Solution:** Integrate a version-vulnerability mapping engine to cross-check versions with known vulnerability databases.

- **Result:** Queries were designed to check for vulnerabilities across specific software versions, ensuring only accurate data was included in the SBOM.

```
class SBOMService:
    def __init__(self, sparql_client):
        self.sparql_client = sparql_client
        self.dependency_regex_patterns = [
            (r".*/([^\s/]+)#(\d+)", r"\1 \2"),
            (r".*/([^\s/]+)#%2A", r"\1 *"),
            (r".*/([^\s/]+)#(.+)", r"\1 \2"),
        ]
        self.vulnerability_regex_patterns = [
            (r".*/([^\s/]+)$", r"\1"),
        ]
        logging.info("SBOMService initialized with SPARQL client.")

    def format_dependency(self, uri):
        # Apply each regex pattern to the dependency URI
        for pattern, replacement in self.dependency_regex_patterns:
            uri = re.sub(pattern, replacement, uri)
        return unquote(uri)

    def format_vulnerability(self, uri):
        # Apply each regex pattern to the vulnerability URI
        for pattern, replacement in self.vulnerability_regex_patterns:
            uri = re.sub(pattern, replacement, uri)
        return unquote(uri)
```

### Mismatched Data from Multiple Sources

**Solution:** Applied **data harmonization** and **preprocessing scripts** to align mismatched product/version data across sources.

- **Result:** Cleaned up and unified data for accurate SBOM entries, eliminating "filler" or irrelevant data.

# SBOM Application (Key Features)

## Live SBOM Analysis Workflow:

1. **User/API Request:**
  - React-based UI allows users to input a product name and fetch SBOM data via a Flask API endpoint.
2. **SBOM Service Processing:**
  - Flask routes the request to SBOMService, which retrieves dependencies and vulnerabilities, then generates the structured SBOM data.
3. **SPARQL Query Execution:**
  - The SPARQLClient queries the Knowledge Graph (using SPARQLWrapper) to collect dependencies and vulnerabilities, leveraging real time data.
4. **Response Formatting:**
  - The data is processed, analyzed, and formatted into a structured JSON SBOM, ready for frontend display or external integration.

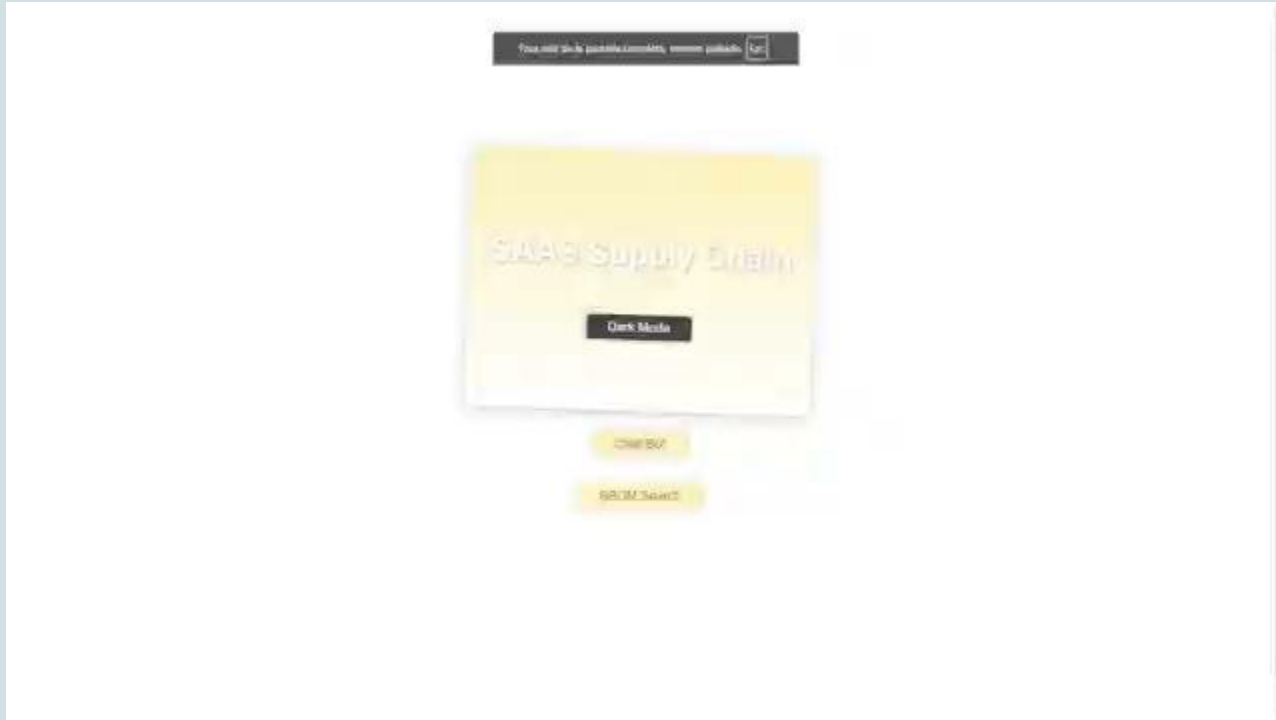
## Frontend:

- **UI (React)**
  - Users input a product name, triggering an asynchronous API call to fetch SBOM data.
- **Dynamic Display**
  - SBOM dependencies, vulnerabilities, and recommendations are displayed in a scrollable list.
- **Asynchronous API Calls**
  - Fetch and display data dynamically with **Axios**; includes error and loading states.

## Backend:

- **Flask API**
  - Routes requests, validates input, and communicates with backend services.
- **SBOMService**
  - Queries SPARQL endpoint for dependencies and vulnerabilities, formats data, and generates SBOM.
- **SPARQLClient**
  - Manages communication with SPARQL endpoint to retrieve product dependencies and vulnerabilities.

# Live DEMO SBOM Analyzer in Action



# Our Unique Advantage (Value)

## 1. Real-Time, Data-Driven Decisions

Most tools give you a snapshot. We give you a live feed. Our SBOM tool pulls real-time data from Purdue's Knowledge Graph, so you're always seeing the freshest, most accurate information about your dependencies and vulnerabilities.

*Continuous, up-to-date insights. No outdated reports.*

## 2. Built for Speed and Scale

We didn't build this for a small project. We built it to scale. Our backend processes massive datasets efficiently, making it ideal for large organizations managing thousands of dependencies. Whether you're a startup or a Fortune 500, our tool grows with you.

*Scalable architecture designed for enterprise-level complexity.*

## 3. Seamless Integration and Usability

No complicated setups. No steep learning curves. Our tools integrate easily into existing workflows and provide results in JSON, ready for audit reports or further analysis. It's plug-and-play security.

*Easy implementation with immediate ROI.*



$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$$
$$= \lim_{h \rightarrow 0} \frac{x^2 + 2xh + h^2 - x^2}{h}$$

$$= \lim_{h \rightarrow 0} \frac{2xh + h^2}{h}$$

$$= \lim_{h \rightarrow 0} \frac{g(x+h) - g(x)}{h}$$
$$= \lim_{h \rightarrow 0} h(2x + h)$$

---

# Natural Language Query System with Chatbot and LLM Integration

---

Prepared with help of chatgpt-4

# ChatBot Overview



## Problem Statement:

- Secure SaaS Supply Chain knowledge graph database built by scholars at Perdue University in RDF (Resource Description Format) which requires SPARQL queries. Users struggle with querying database due to lack of familiarity with SPARQL syntax, limited tools for guiding query formation, and insufficient user-friendly documentation. The challenge is to create a solution that simplifies query creation for non-expert users, ensuring accessibility and ease of use while maintaining the integrity of the RDF data.

## Objective:

- Simplify Information retrieval from RDF datasets using chatbot powered by LLMs.

## Key Components:

- Chatbot for user interaction.
- LLMs for intent detection and query generation.
- Langchain framework for Integrating lower LLM.

# High-Level Architecture

## User Interaction:

- Natural language input via chatbot interface

## Intent Detection:

- identify user intent and extract key entities.

## SPARQL Query Generation:

- Map intent to SPARQL query format.

## Query Parsing:

- Connected to Apache Jena Fuseki

## Response Generation:

- Return human-readable results to user.

[Go Back To Homepage](#)

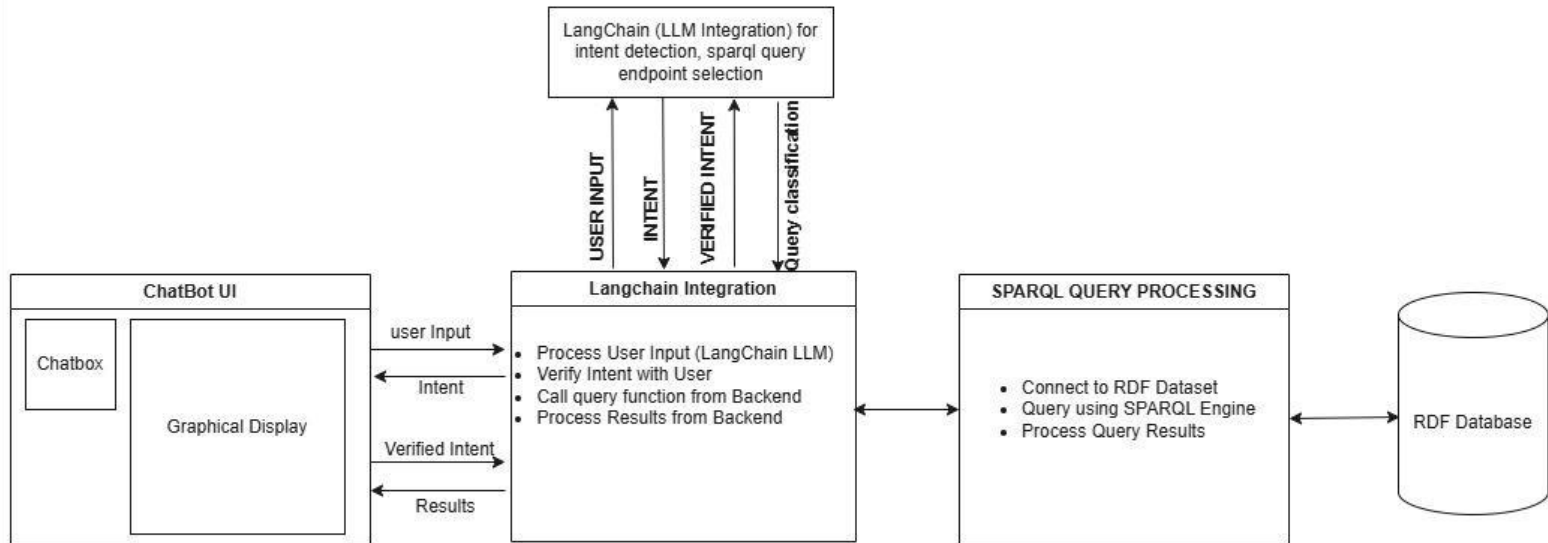
*Note: Currently I'm only programmed to answer dependencies and vulnerabilities for softwares given software name and version.*

Bot: Welcome, what would you like to know?

[Send](#)

No results available.

# Flow Diagram



# NLP Integration

## Functionalities

- **Chatbot:**
  - Custom-built using React DOM
  - Handles user input and output.
- **LLMs :**
  - Detects user intent and entities from natural language.
  - Converts input into structured SPARQL queries.
  - Huggingface Interface to chain LLMs

## Advantages

- **Accessibility:**
  - Allows non-technical users to interact with semantic data easily.
- **Efficiency:**
  - Automates complex query creation.
- **Simplicity:**
  - Leverages LLMs to perform various tasks including typo corrections and other minor fixes which would otherwise require more complex programming efforts

# NLP Integration Challenges/Limitations and Work around

## Lower level LLMs

- **Gpt-2 or similar**
  - Available free of cost
  - Have limited capacity for complex language parsing
- **Current Application Built & Testing Limitations**
  - Gpt-2 model trained sparsely for only two types of queries
  - Ask Bot for dependencies or vulnerabilities for a software
  - SPARQL queries are not generated by LLM
  - LLM selects the predefined query based on intent detection

# NLP Integration Challenges/Limitations and Work around

## Higher level LLMs

- **Gpt-3 or higher**
  - Has advanced NLP capabilities and can generate complex queries correctly with provided ontology description and some training
  - This was verified using ChatGPT in lieu of the GPT-4 API integration (Cost constraints)
- **Current Application Built**
  - Already has functions/Methods to utilize higher LLM built. Change LLM\_type variable to “higher”.
  - Provided API Keys and some training, these functions can be used to automate generation of complex SPARQL queries from user input

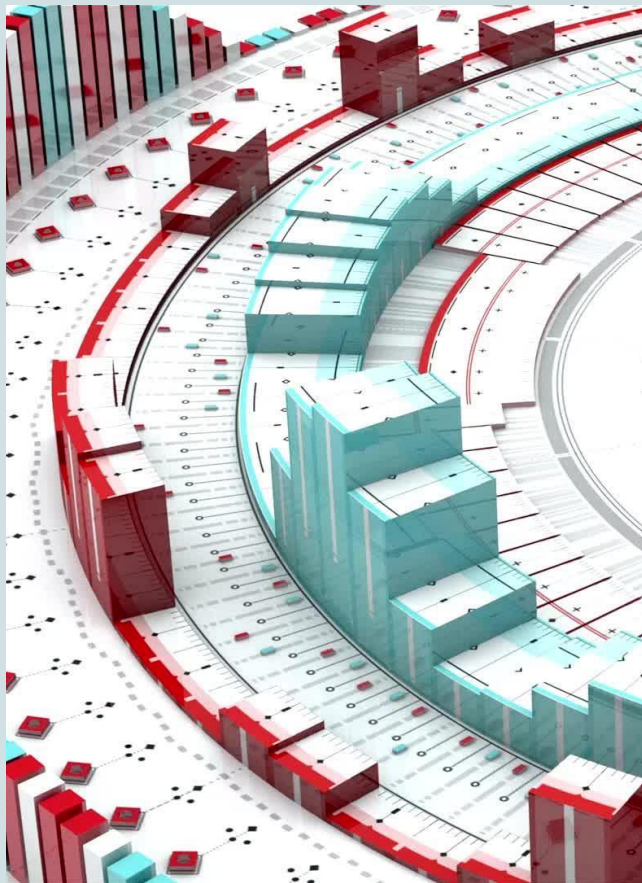
# Advanced Use Cases

## IT Administrator

- **Upgrade software application version**
  - Show the changes in CVEs with the proposed upgrades
- **Dashboard of current system Vulnerabilities and Dependencies**
  - Integration with the SBOM tool
  - Helps track current vulnerabilities and Dependencies in an effective way

## Developers

- **Developer team wants to build a new application XYZ using A,B,C software and H hardware. What are the dependencies and Vulnerabilities associated with it?**
- **What alternative software to be used to minimize CVE of the proposed application**



# ROAD MAP AHEAD

- This was just the first baby step
- Promising and exciting prospect of integrating advanced LLM capabilities to automate data retrieval from RDF dataset
- **Future Work Proposed:**
  - Advanced LLM Integration :  
Train for various query types and more advanced queries
  - Visualization:  
Leverage graph transformers for building interactive graphical visualization

# Thank you for taking the time to review our work!

## To Close:

Our system is built to scale with growing datasets and user demands, handling larger queries and additional data sources seamlessly. The flexible architecture ensures future expansion with minimal adjustments, making it a great tool to utilize for further advancements. The solution greatly improved query performance and data accuracy by optimizing SPARQL queries and ensuring reliable version-vulnerability mapping. Enabling faster, more precise retrieval and aiding tech professionals with tracking dependencies and vulnerabilities more effectively.

*“Any questions or want to discuss?...  
Please don’t hesitate to reach out we look forward to hearing from you”*

### **Megan Mashburn**

Gmail:  
[megandmashburn@gmail.com](mailto:megandmashburn@gmail.com)

LinkedIn:  
<https://linkedin.com/in/megan-mashburn-075598194>

### **Yaser Shafik**

Gmail:  
[contactoyaser@gmail.com](mailto:contactoyaser@gmail.com)

LinkedIn:  
<https://www.linkedin.com/in/yaser-shafik>

### **Ankita Patel**

Gmail:  
[ankitamulticloud23@gmail.com](mailto:ankitamulticloud23@gmail.com)

LinkedIn:  
<https://www.linkedin.com/in/ankita-patel-541183236/>

### **Gerry Murphy**

Gmail:  
[gerry.murphy@pgmode.com](mailto:gerry.murphy@pgmode.com)

LinkedIn:  
<https://www.linkedin.com/in/gerry-m-846123/>

### **Ibrahim Olanigan**

Gmail:  
[ibrahim.olanigan@gmail.com](mailto:ibrahim.olanigan@gmail.com)

LinkedIn:  
<https://www.linkedin.com/in/olanigan/>